NPS52-82-003

NAVAL POSTGRADUATE SCHOOL

Monterey, California

LONG-LIVED TRANSACTIONS - ARE THEY A PROBLEM OR NOT?

Dushan Z. Badal

March 1982

Approved for public release; distribution unlimited

Prepared for:

Naval Postgraduate School
Monterey, California 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

David A. Schrady
Acting Provost

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

This report was prepared by:

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS52-82-003	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: LONG-LIVED TRANSACTIONS - ARE THEY A PROBLEM OR NOT?	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)*: Dushan Z. Badal	8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61152N;RR000-01-10 N0001482AF00001
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93940	12. REPORT DATE: March 1982
	13. NUMBER OF PAGES: 27
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

This report has been submitted for publication. It has been issued as a Research Report for early dissemination of its contents.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Transaction, Transaction model, Database systems, Synchronization, Concurrency control, Transaction processing, Distributed Computing

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This paper discusses three topics. First, we propose a new model of transactions. Second, we discuss long-lived transactions which can last for days or weeks. We describe two real-life examples of such transactions. Third, we discuss concurrency control for such long lived transactions.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

LONG-LIVED TRANSACTIONS - ARE THEY A PROBLEM OR NOT?

Dushan Z. Badal

Naval Postgraduate School

Computer Science Department

Monterey, California 93940

Abstract

This paper discusses three topics. First, we propose a new model of transactions. Second, we discuss long-lived transactions which can last for days or weeks. We describe two real-life examples of such transactions. Third, we discuss concurrency control for such long lived transactions.

1. INTRODUCTION.

The concept of transaction has been recognized, during the last few years, to be useful as an abstraction for structuring some applications such as airline reservations, electronic fund transfers and car rentals. In all of these applications the transactions are simple and they are short lived, i.e., they are short duration transactions. There has been considerable effort in industry to implement transaction processing systems. There has been also considerable theoretical work done in academia and in research establishments on concurrency control. This work deals with the design, correctness, performance, robustness/reliability and complexity of mechanisms which can support the transaction concept [ESW76, BAD78, BAD79, BAD80, BAD80a, BAD81, BER78, BER79, BER81, ATT82, CER82, THO79, ALS76, STO79, STO78, ELL77, LAM76, KUN79, REE78, MOL79, KAN79, LEL78, GRA78, GRA80, FIS82, LYN82, PAP82, REV82, CHE82].

It has been recognized that despite numerous papers on concurrency control mechanisms there seem to be two [GRA81] or three [BAD81] basic classes of concurrency control mechanisms. Gray [GRA81] distinguishes time-domain addressing and locking and logging. Badal [BAD81] besides distinguishing similar classes, also considers an additional class called the MEO class. (Badal's MES or Mutual exclusion over a set is the same as Gray's locking and logging class. Similarly, Badal's S class is analogous to Gray's time-domain addressing). The MEO class is a hybrid class because in some respects it uses the same concepts as both the time-domain addressing and locking and logging. In particular

the MEO class shares with time-domain addressing a notion of unique identifiers (or time stamps) and a notion of multiple (more than two) versions of the same data object. On the other hand, the MEO class shares with the locking and logging approach the notion of logging. The principal idea of the MEO class as proposed in [BAD81] is to use logging, or more precisely the ordering or sequencing of transactions actions in logs, (but not locking and locks or time stamps) for synchronization as well as recovery of transactions having unique ID (or sequence number or time stamp).

The fact that the MEO class concurrency control proposed in [BAD81] uses a concept of multiple data versions seems to provide it with a capability to cope with two difficult and unresolved issues in concurrency control. The first issue is network partitioning and the automatic reconciliation of partitions. The second issue is the support of long-lived transactions such as travel agent, escrow or insurance transactions. In this paper we address the second problem. As pointed out in [GRA81] the present concept and model of transactions and proposed concurrency control mechanisms based on time-domain addressing or locking and logging can not readily, if at all, support long lived transactions which can take days or weeks.

The paper is organized as follows. In section two we discuss in detail the traditional model of transactions and we suggest a new model. In section three we discuss long-lived transactions and in section four we discuss concurrency control for long-lived transactions.

2. TWO MODELS OF TRANSACTIONS.

Before describing two transaction models, we will briefly discuss the concept of transaction. The transaction concept derives from contract law. In making a contract, two or more parties negotiate the contract. Once an agreement among the parties involved is reached, either through direct negotiation or through the independent party, the contract becomes legally binding. Of course, a contract is simply an agreement. Individuals can violate it if they are willing to break the law. But legally, a contract (transaction) can only be annulled if it was illegal in the first place. Adjustment of a bad transaction is done via compensating transactions (including legal redress). Thus, as indicated in [GRA81] the transaction concept emerges with the following properties:

Consistency: the transaction must obey legal protocols.

Atomicity: it either happens or it does not; either all are bound by the contract or none are.

Durability: once a transaction is committed, it cannot be abrogated.

2.1. ONE MODEL OF TRANSACTIONS [GRA81].

Translating the transaction concept to the realm of computer science, we observe that most of the transactions we see around us (banking, car rental, or buying groceries) can be represented in a computer as transformations of a system state.

A system state consists of records and devices with changeable values. The system state includes assertions about the values of records and about the allowed transformations of the values. These

assertions are called the system consistency constraints.

The system provides actions which read and transform the values of records and devices. A collection of actions which comprise a consistent transformation of the state may be grouped to form a transaction. Transactions preserve the system consistency constraints - they obey the laws by transforming consistent states into new consistent states.

Transactions must be atomic and durable: either all actions are done and the transaction is said to commit, or none of the effects of the transaction survive and the transaction is said to abort.

These definitions need slight refinement to allow some actions to be ignored and to account for others which cannot be undone. Actions on entities are categorized as:

Unprotected: the action need not be undone or redone if the transaction must be aborted or the entity value needs to be reconstructed.

Protected: the action can and must be undone or redone if the transaction must be aborted or if the entity value needs to be reconstructed. The result of protected action is usually not visible to the outside world until a transaction commits.

Real: once done, the action cannot be undone.

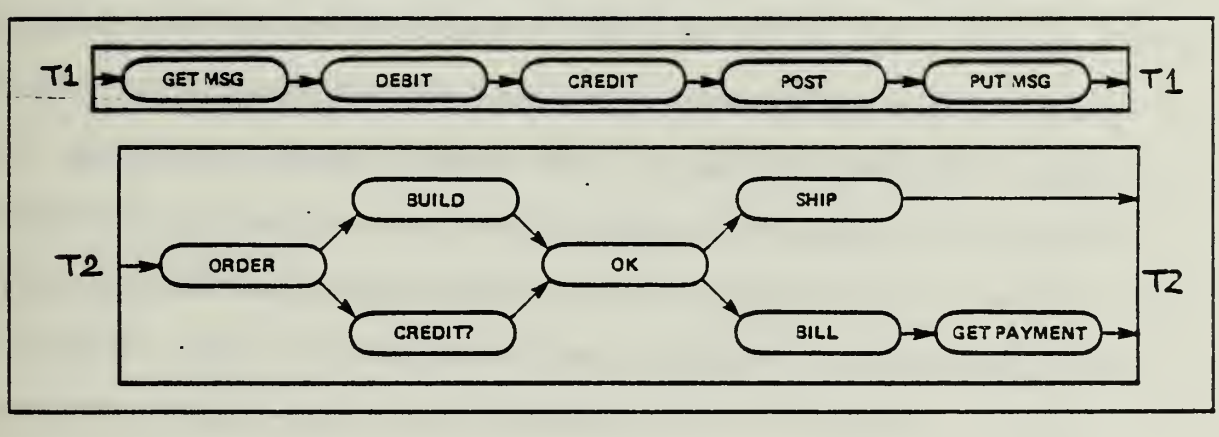
Operations on temporary files and the transmission of intermediate messages are examples of unprotected actions. Conventional database and message operations are examples of protected actions. Transaction commitment and operations on real devices (cash dispensers and airplane wings) are examples of real actions.

Each transaction is defined as having exactly one of two outcomes: committed or aborted. All protected and real actions of committed transactions persist, even in the presence of failures. On the other hand, none of the effects of protected and real actions of an aborted transaction are ever visible to other transactions.

Once a transaction commits, its effects can only be altered by running further transactions. For example, if someone is underpaid, the corrective action is to run another transaction which pays an additional sum. Such post facto transactions are called compensating transactions.

A simple transaction is a linear sequence of actions. A complex transaction may have concurrency within a transaction; the initiation of one action may depend on the outcome of a group of actions. Such transactions seem to have transactions nested within them, although the effects of the nested transactions are only visible to other parts of the transaction (see Figure 1).

Figure 1. Two transactions. T1 is a simple sequence of actions. T2 is a more complex transaction which demonstrates parallelism and nesting within a transaction.



2.2. ANOTHER MODEL OF TRANSACTIONS.

Another way to model a transaction is to consider it as a set of subtransactions, where each subtransaction is a unit of atomicity, recovery and consistency. The composition of subtransactions, which is the transaction, is a composition of units of atomicity, recovery and consistency. Thus, the transaction itself should exhibit atomicity, consistency and recoverability. The subtransaction is defined to be a sequence of read and update actions.

A subtransaction can be a single update or read action. Each subtransaction can make only temporary changes to the database, i.e., each subtransaction can generate only a new temporary version of any data object (DO) that it updates. The temporary versions are visible to and available for updates from other transactions. However, such temporary changes to the database have to be consistent, recoverable and atomic in the sense that all temporary changes either occur or none occurs. The temporary changes to the database become permanent when a transaction commits. A transaction is atomic in the sense that all temporary changes produced by the subtransactions of a given transaction become either permanent or aborted.

Comparing this transaction model with one in section 2.1. we see the following major differences:

1) this model requires that subtransaction is a unit of consistency as opposed to the notion of the transaction being the single unit of consistency

2) this model introduces two types or levels of atomicity. One

level of atomicity deals with generation of temporary DO versions and the other deals with commitment of these versions.

3) this model introduces the notion of temporary DO versions and therefore of temporary database states. The notion of temporary DO version generation corresponds directly to the notion of protected action in the previous transaction model. This means that the temporary DO versions can and must be undone or redone if the transaction is aborted.

We consider the notion of temporary data and database states a crucial one in our transaction model. We have observed several real-life applications, such as escrow and travel agent operations, and we have come to the conclusion that there is a strong notion of temporary data. For example in an escrow process there is a difference between granting a loan to a buyer and executing that loan. The granting of a loan by a bank is a temporary state. If all of the conditions of the escrow are satisfied, then the loan is executed (committed); otherwise it is cancelled (aborted).

3. LONG-LIVED TRANSACTIONS.

The transaction concept was adopted to ease the programming of certain applications. Indeed, the transaction concept is very effective in areas, such as airlines reservation, electronic funds transfer and car rentals, where each application consists of simple transactions of short duration. It appears that the traditional concept of transaction, as represented by the transaction model in section 2.1, has adopted a view that transaction, by definition, is simple and short. This has occurred because the first applications of transaction processing were simple and

short transactions. However, the word "transaction" itself does not in general imply any limitation on the duration of transactions. Thus, there seem to be two unresolved problems. One is to anaylze transactions which are not simple and can last for hours, days or weeks. Gray [GRA81] calls them long-lived transactions. The second problem is how to support the execution of such transactions, i.e., what kind of concurrency control, if any, should be used for such transactions. Solutions to these problems are needed in order to support applications of long-lived transactions. Good examples of such applications are escrow, travel, insurance, government, legal proceedings, electronic mail, etc.

In the remainder of this section we discuss the first problem, i.e., we analyze two examples of long-lived transactions. We address the second problem, i.e., how to support long lived transactions, in section 4.

Consider implementing a travel agent system as discussed in [GRA81]. A transaction in such a system consists of:

1. Customer calls the travel agent giving destination and travel dates.
2. Agent negotiates with airlines for flights.
3. Agent negotiates with car rental companies for cars.
4. Agent negotiates with hotels for rooms.
5. Agent receives tickets and reservations.
6. Agent gives customer tickets and gets credit card number.
7. Agent bills credit card.
8. Customer uses tickets.

Not infrequently, the customer cancels the trip and the agent must undo the transaction.

The transaction concept as described in section 2.1 crumbles under this example. We quote from [GRA81]. "Each interaction with other organizations is a transaction with that organization. It is an atomic, consistent, durable transformation. The agent cannot unilaterally abort an interaction after it completes, rather the agent must run a compensating transaction to reverse the previous transaction (e.g., cancel reservation). The customer thinks of this whole scenario as single transaction. The agent views the fine structure of the scenario, treating each step as an action. The airlines and hotels see only individual actions but view them as transactions. This example makes it clear that actions may be transactions at the next lower level of abstraction."

Let's recast this example in terms of our transaction model discussed in section 2.2. Consider the agent transaction as consisting of subtransactions where each subtransaction represents an interaction with airlines, hotels, etc. Such subtransactions are utilized to make reservations with different organizations. However, the reservation itself is a temporary state which has to be made permanent and legally binding by laking a payment or it is aborted either by the travel agent or the organization, if payment is not made within a certain time window. Such time window can be a few weeks to a few hours before the reservation date. As a matter of fact, in airline reservation systems, once a payment has been made then the reservation data is marked as ticketed and this fact makes the airline legally bound to transport a passenger

within a certain time interval from reservation date. Thus, the travel agent transaction consists of subtransactions which generate temporary data (reservations) which are either made permanent (or legally binding) by an agent billing the customer's credit card (step 7 in the example) or they are aborted. The important points we want to make are that first, the reservation data is temporary and available to other transactions before the travel agent commits the transaction. Second, the reservation subtransactions seem to have two levels of atomicity. One level deals with generation of all temporary data (making reservations) and the second level deals with commitment or abortion of all temporary data. The second level of atomicity depends on the first level, i.e., the second level can take place only after the first one has occurred. In terms of our example the travel agent can commit the transaction, i.e., collect the customer's payment and thus make all temporary reservations permanent, or legally binding, only after all reservations have been made.

Let's consider another example of long-lived transaction - an escrow transaction. Such a transaction consists of:

1. Buyer, seller and possibly real estate agent call title company agent giving him a real estate contract. Such contract contains numerous conditions to be fulfilled before the escrow can be closed. The typical conditions concern financing, price, pest control, insurance, etc.

2. The escrow agent opens the escrow by collecting an initial deposit.

3. The escrow agent investigates public records.

4. The escrow agent investigates property tax records (Steps 3 and 4 are called preliminary title work and clearing the title in escrow agent jargon).

5. Buyer and seller fulfill escrow conditions by obtaining financing, any corrective work, insurance, etc.

6. After escrow fulfillment, both buyer and seller sign escrow.

7. The escrow is closed by entering the transfer of ownership into public records and by disbursing the money involved in the transaction.

We will analyze the escrow transaction. In particular we have chosen step 5 which involves buyer and seller interactions with several institutions. For example the seller has to obtain financing by applying for a loan to a local bank. Such application inevitably triggers a nontrivial set of activities within the bank. We ignore them except the fact that they result in loan being either granted or denied. If the loan is granted then this constitutes one condition of escrow fulfillment. However, from the bank's point of view the act of granting the loan is not a permanent change because, before the loan can be executed, all other escrow conditions must be fulfilled. Only after the escrow is closed, i.e., the escrow transaction is committed, the loan becomes effective and money can be transferred to the seller. At that point the bank's temporary loan data becomes permanent. Obviously the temporary loan data can be and is accessible to the bank. Thus, as in the previous example we can observe two levels of atomicity - one on subtransaction level (all temporary data actions have to occur) and the other one

on transaction level (all temporary data either become permanent or aborted). In this example we have also observed that the escrow data during escrow transaction execution (which typically takes weeks) is accessible to the outside world but in a restricted way, i.e., it is confidential. This raises the general question of the interrelation among synchronization, recovery and security.

4. ON CONCURRENCY CONTROL FOR LONG LIVED TRANSACTIONS.

4.1. LOCKING AND LOGGING VS. LONG LIVED TRANSACTIONS.

Locking and logging as presently used is quite unsuitable for long-lived transactions because of the following reasons. First, it is not feasible to lock a data object for weeks. This would occur if we treated a long-lived transaction as just another transaction. Second, if we consider long-lived transactions as consisting of short transactions then there is a problem of committing such transactions because short transactions commit immediately after their execution and return a result to the long-term transaction. However, when the long-term transaction commits it would recommit or abort its already committed short-term transactions. Obviously this violates the concept of transaction - in particular this violates the notion of transaction durability and atomicity.

In [GRA81] two approaches to handling long lived transactions by locking and logging are described. They involve nested transactions and a lower degree of consistency as follows.

One approach to the handling of long-lived transactions which seems

to offer some help, is to view a transaction as a collection of:

. actions on unprotected objects

. protected actions which may be undone or redone

. real actions which may be deferred but not undone

. nested transactions which may be undone by invoking compensating transactions.

Nested transactions differ from protected actions because their effects are visible to the outside world prior to the commit of the parent transaction.

When a nested transaction is run, it returns as a side effect the name and parameters of the compensating transaction for the nested transaction. This information is kept in a log of the parent transaction and is invoked if the parent is undone. This log needs to be user-visible (part of the database) so that the user and application can know what has been done and what needs to be done or undone. In most applications, a transaction already has a compensating transaction, so generating the compensating transaction (either coding it or invoking it) is not a major programming burden. If all else fails, the compensating transaction might just send a human the message "Help, I can't handle this".

This may not seem very satisfying, but it is better than the entirely manual process which is in common use today. At least in this proposal, the recovery system keeps track of what the transaction has done and what must be done to undo it.

At present, application programmers implement such applications using a technique called a "scratch-pad" (in IMS) and a "transaction work area" in CICS. The application programmer keeps the transaction state (his own log) as a record in the database. Each time the transaction becomes active, it reads its scratchpad. This re-establishes the transaction state. The transaction either advances and inserts the new scratchpad in the database or aborts and uses the scratchpad as a log of things to undo. In this instance, the application programmer is implementing nested transactions. It is a general facility that should be included in the host transaction management system.

Some argue that nested transactions are not transactions. They do have some of the transaction properties:

Consistent transformation of the state

Either all actions commit or are undone by compensation

Once committed, cannot be undone

They use the BEGIN, COMMIT and ABORT verbs. But they do not have the property of atomicity. Others can see the uncommitted updates of nested transactions. These updates may subsequently be undone by compensation.

The second approach to the handling of long-lived transaction by locking is to accept a lower degree of consistency [GRA80] so that only "active" transactions (ones currently in the process of making changes to the database) hold locks. "Sleeping" transactions (travel arrangements not currently making any updates) will not hold any locks. This will mean that the updates of uncommitted transactions are visible to other transactions. This in turn means that the UNDO and REDO

operations of one transaction will have to commute with the DO operations of others (i.e., if transaction T1 updates entity E and then T2 updates entity E and then T1 aborts, the update of T2 should not be undone). If some object is only manipulated with additions and subtractions, and if the log records the delta rather than the old and new value, then UNDO and REDO may be made to commute with DO. IMS Fast Path uses the fact that plus and minus commute to reduce lock contention. No one knows how far this trick can be generalized [GRA81].

4.2. ANOTHER SOLUTION TO LONG LIVED TRANSACTION SYNCHRONIZATION.

The transaction model described in section 2.2. is used in a concurrency control proposal described in [BAD81, BAD79, MCE82]. The principal ideas of this proposal can be described as follows. Each data object (DO) in the database has a log associated with it. The data object log contains a history of all actions on a given data object (DO). Transactions are in three possible states - executing, committed or aborted. Similarly, a transaction's entries into DO logs are in three possible states - temporary, committed (permanent) or aborted. DO log entries are created after the transaction executes on a given DO. When there are n updates on a given DO, then there are created n versions of that DO. The i-th version of DO is created by updating the (i-1)-th version. All such versions are temporary and the i-th version can commit, or become permanent only after the (i-1)-th version has committed. However, if the i-th version is aborted then all j versions, j>i, must be aborted as well. In some sense this concurrency control uses a look-ahead technique by allowing a precomputation of DO versions.

All DO versions are available to all transactions at any time during transaction execution. This is an optimistic concurrency control in that it assumes a low frequency of conflicts among transactions and it recognizes the fact that not all conflicts result in an inconsistent database state, i.e., in nonserializable execution. This concurrency control mechanism uses a simple set of rules about the sequencing or ordering of DO log entries to detect and resolve any nonserializable execution of conflicting transactions. It might seem that this concurrency control is subject to a so-called domino effect when one transaction abort triggers the abort of all other transactions which either read or updated its output. Yes, that it is true, but, since this concurrency control is intended for applications with a low frequency of conflicts then the domino effect (which is determined by a frequency of conflicts) should not be significant. Moreover, the concurrency control mechanism as proposed in [BAD81] will decrease and probably eliminate all conflicts among transactions because the detection of nonserializable execution is based on the smallest possible granularity, i.e., on accessed record field (as the DO log entry by a given transaction is created after the transaction' read or update of DO has been executed). This concurrency control uses multiple DO versions which can either be kept indefinitely or they can be deleted after the transaction has committed - that is a matter of choice and available storage technology. If multiple versions of DO's are kept indefinitely and the transaction ID used is a time stamp then this mechanism can support queries using time reference as for example "what were the values of DO's at time t".

We want to point out that since time stamps are not used for synchronization, this concurrency control can use approximately synchronized real time clocks for generation of time stamps - if one wishes to use time stamps as transaction ID's.

The long-lived transactions are treated in this concurrency control mechanism in the same may as short-lived transactions. This means that subtransactions of a (short or long lived) transaction generate temporary versions of data which are accessible to other transactions. When the transaction commits the temporary data versions become permanent or aborted. Since, in our opinion, long-lived transactions generate mostly temporary data, then we seem to be able to handle such transactions in a very natural way. More details on the concurrency control discussed in this section can be found in a forthcoming report [MCE82].

5. CONCLUSIONS.

In this paper we have discussed the problem of long-lived transactions. The main contribution of this paper is in presenting another point of view on, and hopefully some insights into, transaction models, long- lived transactions and concurrency control for long-lived transactions.

6. ACKNOWLEDGEMENTS.

The author wishes to express his appreciation to Bill McElyea for discussing with him many ideas in this paper and for contributions to the presentation.

REFERENCES

[ALS76] Alsberg, P. et al. "Multi-Copy Resileincy Techniques," Center for Advanced Computation, Report CA 6202, University of Illinois, Urbana-Champaign, May 1976.

[ATT82] Attar, R., Bernstein, P. A., and Goodman, N., "Site Initialization, Recovery, and Back-Up in a Distributed Database System," Proc. of the 6th Berkeley Workshop on Distributed Data Management Computer Networks, Asilomar, February 16-19, 1982.

[BAD78] Badal, D. Z., and Popek, G. J. "A Proposal for Distributed Concurrency Control for Partially Replicated Distributed Databases," Proc. of the 3rd Berkeley Conference on Distributed Data Management and Computer Networks, August 1978.

[BAD79] Badal, D. Z., "Correctness of Concurrency Control and Implications in Distributed Databases," Proc. of COMPSAC 79, Chicago, November 1979.

[BAD80] Badal, D. Z., "On the Degree of Concurrency Provided by Concurrency Control Mechanisms for Distributed Databases," Proc. of the Inter. Symposium on Distributed Databases, Paris, France, March 1980.

[BAD80a] Badal, D. Z., "The Analysis of the Effects of Concurrency Control on Distributed Database System Performance," Proc. of the 6th Intern. Conference on Very Large Data Bases, Montreal, October 1980.

[BAD81] Badal, D. Z., "Concurrency Control Overhead or Closer Look at Blocking vs. Nonblocking Concurrency Control Mechanisms," Proc. of the 5th Berkeley Conference on Distributed Data Management and Computer Networks, San Francisco, February 1981.

[BER78] Bernstein, P. A., et al., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases," IEEE Transactions on Software Engineering 4, 3 (May 1978).

[CER82] Ceri, Stefano and Cwicki, Susan "On the Use of Optimistic Methods for Concurrency Control in Distributed Databases," Proc. of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Asilomar, February 16-19, 1982.

[CHE82] Cheng, Wing Kai and Belford, Geneva G., "The Resiliency of Fully Replicated Distributed Databases. Proc of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Asilomar, February 16-19, 1982.

[ELL77] Ellis, C., "A Robust Algorithm for Updating Duplicate Databases," Proc. of the 2nd Berkeley Workshop on Distributed Data

Management and Networks, May 1977.

[ESW76] Eswaran, K. P., et al., "The Notions of Consistency and Predicate Locks in a Database System," CACM 19, 11 (November 1976).

[FAI81] Faissol, S., "Operation of Distributed Database Systems Under Network Partitions," Ph.D. dissertation, UCLA Dept. of Computer Science, July 1981.

[FIS82] Fischer, M. J. and Michael, A., "Sacrificing Serializability to Attain High Availability of Data in an Unreliable Network", Proc. of ACM Symposium on Principles of Database Systems, Los Angeles, March 29-31, 1982.

[GEL79] Gelenbe, E. and Sevcik, K., "Analysis of Update Synchronization for Multiple Copy Data Bases," IEEE Transactions on Computers 28, 10 (October 1979).

[GRA78] Gray, J., "Notes on Database Operating Systems," IBM Research Report RJ 2188, February 1978.

[GRA80] Gray, J., "A Transaction Model," in Automata, Languages and Programming, Lecture Notes in Computer Science 85, Springer-Verlag, 1980.

[GRA81] Gray, J., "The Transaction Concept: Virtues and Limitations," TANDEM TR 81.3, June 1981.

[HAM78] Hammer, M. and Shipman, D., "An Overview of Reliability Mechanisms for a Distributed Data Base System," Spring Compcon 78, February 28 - 3 March, San Francisco, pp. 63-65.

[HER79] Herman, D., et al., "An Algorithm for Maintaining the Consistency of Multiple Copies," ibid KUN79.

[KUN79] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," Proc. of VLDB Conference, Rio De Janeiro, Brazil, October 1979.

[LAM78] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," CACM 21, 7 (July 1978). March 1976.

[LEL78] LeLann, G., "Algorithms for Distributed Data-Sharing Systems Which Use Tickets," ibid BAD78.

[LID79] Lindsay, G. B., et al., "Notes on Distributed Databases," IBM Research Report RJ 2571, July 1979.

[LIN79] Lin, W. K., "Concurrency Control in a Multiple Copy Distributed Database System," ibid BAD78.

[LYN82] Lynch, N., "Multilevel Atomicity," Proc. of ACM Symposium on Principles of Database Systems, Los Angeles, March 29-31, 1982.

[MCE82] McElyea, W. P., "The Nonblocking Concurrency Control for Distributed Databases", Computer Science Department, Naval Postgraduate School, M.S. Thesis, June 1982.

[MIN78] Minoura, T., "Maximally Concurrent Transaction Processing," ibid BAD78.

[MOL79] Garcia-Molina, H., "Performance of Update Algorithms for Replicated Data in a Distributed Database," Ph.D. dissertation, Dept. of Computer Science, Stanford University, June 1979.

[PAP79] Papadimitriou, C. M., "Serializability of Concurrent Database Updates," JACM 26, 4 (October 1979).

[PAR81] Parker, D. S., Popek, G. P., Rudisin, G., et al., "Detection of Mutual Inconsistency in Distributed Systems," Proc. 5th Berkeley Workshop on Distributed Data Management and Computer Networks, February 1981.

[PAR82] Parker, D. Scott and Ramos, Raimundo A., "A Distributed File System Architecture Supporting High Availability," Proc. of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Asilomar, February 16-19, 1982.

[REE78] Reed, D. P., "Naming and Synchronization in Decentralized Computer Systems," MIT/LCS/TR-205, MIT, Laboratory for Computer Science, September 1978.

[REU82] Reuter, A., "Concurrency on High-Traffic Data Elements," Proc. of ACM Symposium on Principles of Database Systems, Los Angeles, March 29-31, 1982.

[RIE79] Ries, D. R., "The Effects of Concurrency Control on Database Management System Performance," Ph.D. dissertation, Computer Science Dept., University of California, Berkeley, April 1979.

[ROS78] Rosenkrantz, D. J., et al., "System Level Concurrency Control for Distributed Database Systems," ACM TODS 3, 2 (June 1978).

[STO78] Stonebraker, M., "Concurrency Control of Multiple Copies of Data in Distributed INGRES," ibid BAD78.

[STO79] Stonebraker, M., "Concurrency Control of Multiple Copies of Data in Distributed INGRES," IEEE Trans. on Software Engineering, Vol. SE-5, 3, 188-194 (May 1979).

[THO79] Thomas, R., "A Solution to the Concurrency Control Problem for

Multiple Copy Databases," ACM TODS 4, 2 (June 1979).

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93940	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1
Chairman, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, CA 93940	40
D. Z. Badal, Code 52Zd Department of Computer Science Naval Postgraduate School Monterey, CA 93940	5
Robert B. Grafton Office of Naval Research Code 437 800 N. Quincy Street Arlington, VA 22217	1
David W. Mizell Office of Naval Research 1030 East Green Street Pasadena, CA 91106	1